# CABINS : Case-Based Interactive Scheduler

Kazuo Miyashita
miyashita@cs.cmu.edu

Katia Sycara
katia@cs.cmu.edu

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

## 1. Introduction

Although there has been a lot of progress in knowledge-based scheduling [5, 4], there is still a need for schedule improvement and repair through interaction with a human scheduler. There are several reasons for this. First, a user's preferences on the schedule are context dependent (e.g., may depend on the state of the scheduling environment at a particular time). Also, interactions among preferences and effective tradeoff very often depend on the particular schedule produced. This means that generally a user of the scheduling system can't fully specify his/her preferences a priori before getting the scheduling results from the system. By looking over the obtained schedule results, the user often thinks of additional preferences. Consider, for example a situation where a human scheduler does not like to use MACHINE-A which is substitutable for MACHINE-B but is of lower quality than MACHINE-B for processing ORDER-X. The reason high quality results are desired is that ORDER-X belongs to a quite important client. Suppose, however, that the schedule indicates that ORDER-X is tardy by an amount above an acceptable tardiness threshold due to demands on MACHINE-B (by orders more important than ORDER-X). Then, the human scheduler may decide to use the less preferable machine, MACHINE-A for the less important order, ORDER-X. If the tardiness was below the threshold, he/she may prefer to allow a tardy order. It is very difficult to elicit this type of preference and preference thresholds from the human scheduler independent of the presence of a particular context.

The second reason interactive schedule repair is desirable is that it is impossible for any given knowledge based scheduling model to include all the constraints that may be relevant. Current advanced scheduling systems can exploit very complicated models to represent the factory, orders and user's preferences. But no matter how richly the model is constructed, there are always additional factors which may influence the schedule but had not been represented in the model. For example, for a certain foundry it may be good to decrease usage of a sand casting machine during the summer, because the combination of heat and humidity of the weather may make it slower than usual. But how should the model of the scheduling system represent the season, weather or humidity? And isn't it necessary for the model to represent time of the day, strength of wind or health of a machine operator and so on? [2]. Nevertheless these factors, that an experienced human scheduler learns to take into consideration, could have a big influence on schedule quality but it is very difficult to represent in a principled manner so they can be used by an automated scheduling system.

The third reason interactive schedule repair is desirable is that factories are dynamic environments. Unexpected events, such as operator absence, power failure and machine breakdowns frequently happen. Therefore, it is necessary for the scheduling system to adapt to the events in the factory environment as soon as possible by reactively repairing the existing schedule. Although initial progress has been made in automatic schedule repair [3], human intervention may be necessary as a result of the reasons given (context dependent user preferences, and difficulty of representing all relevant constraints).

Another consequence of the above is that *local repair rather than re-scheduling* is more desirable, since re-scheduling will suffer from the same ills as the initial scheduling. In addition, it is in general desirable [3] to minimize disruption to the shop floor. If re-scheduling from the point of failure is attempted, the new schedule may be drastically different from the original schedule, thus necessitating disruption of the work flow in the shop, and new work allocation. The new schedule, moreover may solve the current problem but introduce new problems that have to be solved.

One extremely beneficial side effect of interactive schedule repair is the insight that the user obtains into his/her scheduling preferences and their context of applicability. The process of interactive repair requires the human scheduler to analyze the current problem, repair it by clarifying or modifying his/her preferences and finally evaluate the result. This gives the human scheduler good opportunities to understand his/her criteria in diverse situations. So later when he/she encounters a problem that is similar to a previous one, he/she can be reminded of the applicable previous repair and re-use it in the current situation.

### 1.1. Why case-based repair?

Case-based Reasoning (CBR) is a recent AI problem solving paradigm [1]. A CBR system tries to solve a problem by (1) retrieving the most similar case with the current problem from its case base, (2) modifying it to adapt to the current situation and (3) applying it to the current problem. At the end of problem solving, the new solved problem is stored as a new case in the case memory. As a computational model the first feature of CBR is its method of knowledge acquisition. In CBR the unit of knowledge is the case, which is an experience encountered during problem solving. This makes it easier to articulate, examine and evaluate the knowledge. The second feature is its learning capability. A CBR system can remember its performance and modify its behavior to avoid repeating prior mistakes. The third feature is its adaptive power. By reasoning from analogy with the past experiences, a CBR system should be able to construct solutions to novel problems. These features make CBR very attractive for interactive schedule repair.

Because a case describes a particular specific experience, the factors that were deemed relevant to this experience can be recorded in the case. This description fully captures the dependencies among features and their context. So if a similar situation is encountered, the system can re-use the repairing method which is stored in the retrieved case. In addition, a case serves as a knowledge structuring mechanism so that all relevant factors are local to a case rather than distributed through the system (as happens with rule based systems). Even when the result of applying the repairing method of the retrieved case turns out to be failure, if the user can explain the failure, then the system can create a new case based upon this failure experience and store it as a new case along with the associated explanation. Thus, as the case base is enriched with successful and failed experiences, the system becomes more robust for various type of schedule defects that would have been difficult to predict in advance. This enables the replacement of expert users with novices that rely on the system's experiences.

## 2. Case-based Interactive Scheduler (CABINS)

Based upon the above discussion, we are developing the Case-based Interactive Scheduler (CABINS) whose goal is to support interactive schedule repair. A CABINS user is envisioned to be a person who is responsible for making schedules in advance of production. In making an initial schedule, the user may be assisted by an automated scheduling system. If the user identifies undesirable features of the current schedule, he/she uses CABINS for schedule repair, so as to improve the current schedule. CABINS finds defects in scheduling results and repairs them by patching locally or modifying part of its model (resources, orders, shifts and user's preferences).

### 2.1. System Architecture

After the initial schedule is made, it is examined by the user and the defect detector (a rule-based system) to find undesirable parts in the existing schedule. If some defects are detected, the information about the defects are passed to the repairer. If local repairing is determined to be feasible by the repairer, resource reservations in the current schedule are directly modified or canceled by the repairer and the scheduler is asked to re-schedule the conflicting operations whose reservations were canceled. When local repair turns out to be impossible, the repairer modifies the scheduling model and re-scheduling is attempted based on the modified model. The overall goal of CABINS is to make repairs as cheap as possible trying at the same time to minimize interfering side effects of these repairs on the current schedule. Figure 2-1 depicts the architecture of CABINS.

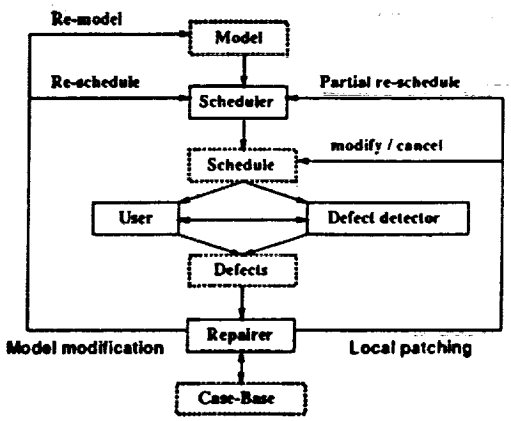


**Figure 2-1:** Architecture of CABINS

### 2.2. Schedule Repairing Process

The processing of CABINS has four stages:

- defect detection
- defect selection
- selection of repair strategy
- selection of repair tactics

Currently defect detection and defect selection are performed by the user who finds the most important defect and identifies the features associated with the defect. These features are used as indices into the case memory to find similar past defects. Out of the retrieved similar past defects, the *least critical* is selected. To determine defect criticality, the system uses the cost of repairing the defect as a measure: the lower the repair cost, the less critical the defect. Low repair cost is usually associated with local patching whereas high cost means that more changes are made to the overall schedule. So, beginning with the lowest cost repair is a good heuristic since the defect can be potentially fixed cheaply.

CABINS uses two level of repairs: repair strategies and repair tactics. A repair strategy is associated with a particular high level description of classes of defects. Each repair strategy has a variety of repair tactics associated with

it. The repair tactics are appropriate for particular specializations of the defect classes. We have identified two general types of repair strategies: local patching and model modification.

To select a strategy for repairing important defects, CABINS looks for the most similar case to the current situation in the case base and selects the same strategy which succeeded in the past case. The system has several alternative strategies for each defect and one of them is selected based on the feature similarity of the current situation and the past experience. Some of the features that we are currently using for case retrieval are various defect types, such as order tardiness and various schedule characteristics, such as schedule tightness, inter-order slack, and machine idle time. For example, if the type of defect is "tardy order", there are seven repair strategies:

1. Reduce the slack between operations in the tardy order
2. Reduce the idle-time of resources needed by operations in the tardy order
3. Relax due-date constraint of orders (the tardy order or interfering orders)
4. Relax release-date constraint of orders (the tardy order or interfering orders)
5. Reduce the shop load
6. Increase shifts
7. Increase resource capacity.

The first two strategies belong to the general category "local patching" and the rest to the category "model modification".

In general, we have presented the repair strategies in order of expensiveness (from the cheaper --strategy 1 to most expensive --strategy 7). For tardiness repair, the discriminating feature between selecting cases with repair strategies in classes 1 to 2 and selecting cases with repair strategies 3 to 7 is the tightness of the current schedule. If the current schedule is not very tight (i.e., there are a lot of idle intervals on resources needed by operations of the tardy order), CABINS will select cases where tardiness was repaired by local patching. Whether cases with repair-strategy-1 or repair-strategy-2 will be selected depends on whether, beside enough idle interval, there is also slack between adjacent operations of the tardy order. If there are, then cases where strategy-1 was used will be selected. Tactics associated with strategy-2 could be to move every operation of the tardy order upstream (left shifting) on the time line if enough idle interval is available for the operation.

If the current schedule is tight, then cases that prescribe model modification rather than local patching will be retrieved. If there are no discriminating features to determine the applicability of strategies 3 to 7, CABINS uses the default ordering: use strategies in ascending cost. The cheapest model modification is relaxing due-date constraints of the tardy order or interfering orders (strategy-3). This is cheap since it is easily accomplished and has no side effects on the shop floor environment. On the other hand, reducing the factory load (strategy-5) (e.g., by subcontracting orders) and re-scheduling is in general more expensive than relaxing due dates of interfering orders because one must determine the orders to be subcontracted out, price of subcontracting, possible delays etc. An additional concern is that the resulting schedule might not be entirely satisfactory and may need to be repaired anew. Similarly, strategies 6 and 7 are increasingly expensive, since additional investments in paying overtime or buying new machines are needed.

Although strategy-3 is the cheapest of the repair strategies of type "model modification", it may not always be desirable. To determine applicability of strategy-3, CABINS retrieves cases where application of strategy-3 has failed. If other features of the current situation match features of the past failures of strategy-3 (e.g., the tardy order has a stiff penalty for tardiness), then CABINS is warned that strategy-3 is not applicable. Similarly, if there are no discriminating features to distinguish among the application of strategies 4 to 7, retrieval of previous cases where the strategy under consideration has failed gives the system additional discriminating information. Thus, CABINS uses the default ordering of repair strategies as well as successful case application as necessary conditions of the applicability of particular repairs; it uses past failures as sufficiency conditions. As more cases are encountered, both the necessary and sufficiency conditions are refined. Therefore, it is hoped that CABINS can improve its performance over time.

For each repair strategy, there could be a variety of repair tactics that are applicable. For repairing order tardiness, there is a variety of appropriate tactics for local patching. Below, we present some of these tactics.

1. left-shift on same resource: move the operation as much to the left as possible, while maintaining the amount of disruptions as small as possible.
2. left-shift on substitutable resource: if the operation that is desired to be moved has a substitutable resource, then move the operation as much to the left as possible, while maintaining the amount of disruptions as small as possible.
3. swap on same resource: find another operation which is to the left of the operation to be moved on the same resource and whose duration is approximately equal to the duration of the current operation and swap the two operations.
4. swap on substitutable resource: if the operation that is desired to be moved has a substitutable resource, then find another operation which is to the left of the operation to be moved on the substitutable resource and whose duration is approximately equal to the duration of the current operation and swap the two operations.

The last two tactics may result in tardiness of other orders but this may be allowable.

For model modification, possibly applicable tactics along with the associated repair strategy are:

1. relax-due-date-of-tardy-order (strategy-3)
2. find-most-interfering-order with the current tardy order and make it tardy (strategy-3)
3. relax-release-date-of-tardy-order (strategy-4)
4. find-most-interfering-order with the current tardy order and make it start earlier (strategy-4)
5. subcontract-least-profitable-order to create more slack (strategy-5)
6. subcontract-most-interfering-order to create more slack (strategy-5)
7. overtime-work on weekday (2 hours) (strategy-6)
8. overtime-work on weekend (8 hours) (strategy-6)
9. increase-capacity-of-most-critical-resource (strategy-7)
10. capacity-of-substitutable-resource-of-most-critical-resource (strategy-7)

Each retrieved case has been repaired by possibly using a combination of repair strategies and tactics. Upon recognition of similarities in schedule defects and defect context, the appropriate repair plan could be applied. If the application of a repair step leads to failure, the user is asked to supply a possible explanation of the failure. The failure is then stored in memory so it can be retrieved and help the user avoid similar failures in the future.

## 3. Example

In this chapter we explain how CABINS works by using a simple example. In the example we make a schedule of 4 orders on 5 resources. Each order has a client, fixed release-date and fixed due-date. Every order is composed of 5 operations (ope-1 to ope-5), which should be ordered in that order. Each operation has fixed duration and requires one resource which may or may not have a substitutable resource. The detail specifications of the example problem are depicted in figure 3-1. In Figure 3-2 we show the result of the original scheduling. Each rectangle represents the reservation of each operation over the time-interval on the machine. The small number inside each rectangle shows the order to which the operation belongs. In scheduling the 4 orders, the scheduler failed to meet the due-date of order-3 by 130. (The due-date of order-3 is 790, while order-3 is scheduled to finish on 920.) Suppose that the client of order-3 has had the late shipment of his orders several times, s/he is sure to cancel her/his contract as a result of our more tardy shipment. Therefore, finding and fixing this situation is critical. A human scheduler at the factory tries to fix this problem by consulting with CABINS.

First, CABINS considers the current problem as a case by compiling the current scheduling results with respect to the tardiness of order-3. A human scheduler gives additional contextual information to it if s/he finds it's necessary or helpful for finding the solution of the current problem. The vocabulary of this information is maintained by CABINS and a human scheduler can update it by adding/deleting terms. Figure 3-3 shows the contents of this example problem case.

Then, CABINS tries to retrieve the case most similar

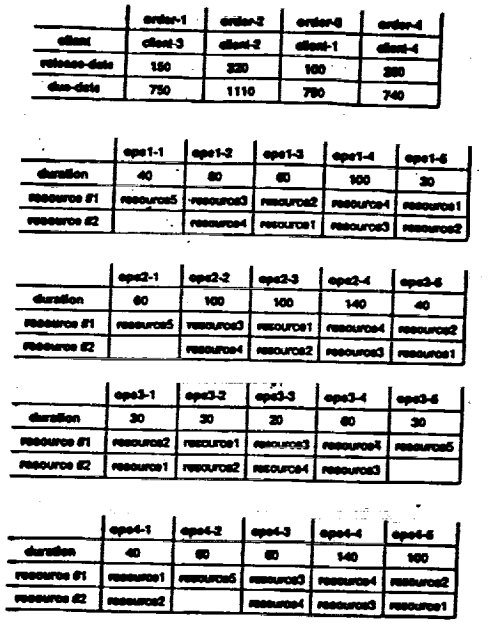

**Figure 3-1:** Problem Specifications

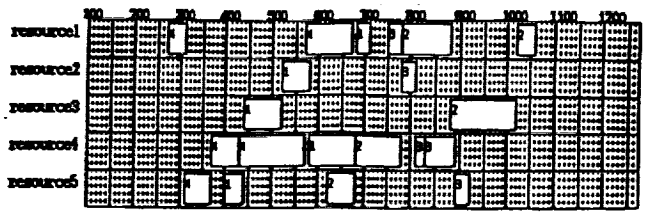

**Figure 3-2:** Initial Schedule

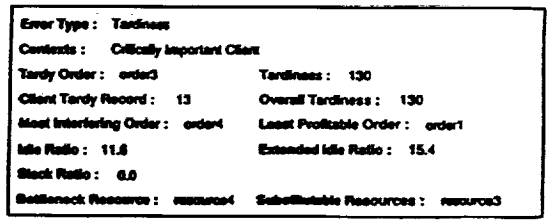


**Figure 3-3:** Current Problem Case

cases to the current problem case from its case-base library. The retrieved case includes not only the problem situation description but also repairs and repair outcomes. For repair strategy selection, every solution includes the information of the selected strategy, the result of applying the strategy and the explanation of why it succeeded or failed. The explanation of the solution outcome is added to the case by a human scheduler only when s/he thinks it is necessary for credit or blame assignment of the selected strategy. Figure 3-4 depicts the retrieved case to solve this example problem.

After display of the retrieved cases, a human scheduler examines whether s/he can apply the same solution method

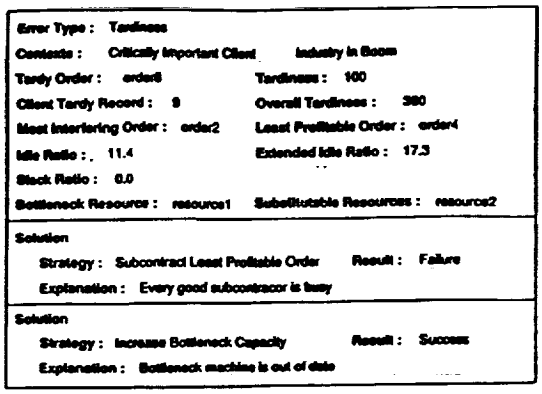


**Figure 3-4:** Retrieved Case

to the current problem. Even when the result of the solution in the retrieved case was failure, the solution may be worth trying if the explanation of failure given in the previous case does not hold in the current situation. On the other hand, a human scheduler should also check the validity of the explanation of a successful previous solution before s/he applies it to the current problem. In this example, even though the first solution failed when it was applied in the precedent case, a human scheduler can try to apply it, because the explanation of the failure given ("Every good subcontractor is busy") is apparently related to the description of the context of the problem ("Industry in Boom"). Therefore the explanation is not necessarily true in the current situation which doesn't share the same context. Note that those judgments are done by a human scheduler. However, by retrieving and displaying previous similar cases, CABINS gives her/him useful information to help making her/his decision. Moreover, the greater the number of new cases that are added into the case-base library, the more likely CABINS is to retrieve the case which is close enough to the current problem. Therefore, it becomes progressively easier through CBR to decide whether the solution of the retrieved case is applicable or not.

After determining the solution method, a human scheduler can execute it by interacting with the scheduling system. Figure 3-5 depicts the result of rescheduling order-3 after subcontracting the least profitable order (order-1) in this example. It shows that order-3 meets its due-date, i.e. the repair was successful.

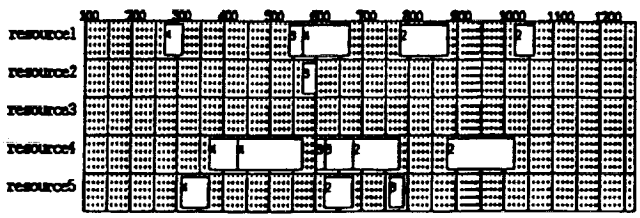


**Figure 3-5:** Repaired Schedule

## 4. Concluding Remarks

In this paper we discuss the need for interactive factory schedule repair and improvement, and identify case-based reasoning (CBR) as an appropriate methodology. Case based reasoning is the problem solving paradigm that relies on a memory for past problem solving experiences (cases) to guide current problem solving. Cases similar to the current case are retrieved from the case memory, and similarities and differences of the current case with past cases are identified. Then a best case is selected and its repair plan is adapted to fit the current problem description. If a repair solution fails, an explanation for the failure is stored along with the case in memory, so that the user can avoid repeating similar failures in the future.

So far we have identified a number of repair strategies and tactics for factory scheduling and have implemented a part of our approach in a prototype system, called CABINS. As a future work, we are going to scale up CABINS to evaluate its usefulness in a real manufacturing environment.

## References

[1] Kolodner, J., Simpson, R. and Sycara, K.
A Process of Case-Based Reasoning in Problem Solving.
In *Proceeding of the Ninth International Joint Conference on Aritificial Intelligence*, pages 284-290. IJCAI, Los Angeles, CA, 1985.

[2] K.Mckay, J.Buzacott, F.Safayeni.
The Scheduler's Knowledge of Uncertainty: The Missing Link.
In *Proceedings of IFIP Working Conference on Knowledge Based Production Management Systems*. Galway, Ireland, 1988.

[3] P.S. Ow, S.F.Smith, A.Thiriez.
Reactive Plan Revision.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77-82. AAAI, St-Paul, Minnesota, 1988.

[4] Norman Sadeh.
*LOOK-AHEAD TECHNIQUES FOR MICRO-OPPORTUNISTIC JOB SHOP SCHEDULING*.
PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[5] Stephen F.Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin Dirk C.Matthys.
AN INTEGRATED FRAMEWORK FOR GENERATING AND REVISING FACTORY SCHEDULES.
*Journal of the Operational Research Society* , 1990.